Surya Group OF Institution

Naan Muthalvan

IBM

|  |
| --- |
| **IBM - Artificial Intelligence**  **PACHAIAMMAL P V**  **422221104023**  **Team - 05** |

Sentiment analysis for marketing

Sentiment analysis is a technique that uses natural language processing and machine learning to analyze the emotions and opinions expressed in text data. It can help marketers to understand how their customers feel about their products, services, or brand, and to improve their marketing strategies accordingly.

There are many ways to perform sentiment analysis using Python, one of the most popular programming languages for data science.

Some of the possible methods are:

* Using pre-trained models from Hugging Face, a platform that provides a large collection of state-of-the-art natural language processing models. You can use the pipeline function from the transformers library to create a sentiment analysis pipeline that can take any text input and return a label (positive or negative) and a score (confidence level).

pip install -q transformers

from transformers import pipeline sentiment\_pipeline = pipeline("sentiment-analysis") data = ["I love you", "I hate you"] sentiment\_pipeline(data)

* Using the TextBlob library, which is a simple and intuitive way to perform sentiment analysis in Python. It has a built-in sentiment analyzer that returns a polarity (between -1 and 1) and a subjectivity (between 0 and 1) score for any text input.

pip install -q textblob

from textblob import TextBlob data = ["I love you", "I hate you"] for text in data:

blob = TextBlob(text) print(blob.sentiment)

* Using the NLTK library, which is a comprehensive toolkit for natural language processing in Python. It has various modules and resources for sentiment analysis, such as the VADER (Valence Aware Dictionary and sEntiment Reasoner) tool, which is a lexicon-based approach that uses a list of words with predefined sentiment scores to calculate the overall sentiment of a text input.

pip install -q nltk import nltk

nltk.download('vader\_lexicon')

from nltk.sentiment.vader import SentimentIntensityAnalyzer sia = SentimentIntensityAnalyzer()

data = ["I love you", "I hate you"] for text in data:

print(sia.polarity\_scores(text))

* Building your own sentiment analysis model using TensorFlow, which is an open- source framework for machine learning and deep learning. You can use the keras API to create a neural network model that can learn from labeled text data and predict the sentiment of new text inputs.

**Table of Contents:**

* [How to Perform Sentiment Analysis in Python?](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)

1. [Python Pre-Requisites](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)
2. [Reading the Dataset](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)
3. [Data Preprocessing](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)
4. [TF-IDF Transformation](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)
5. [Building and Evaluating the ML Model](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)
   * [Sentiment Analysis with Python: Next Steps](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)

[How to Perform Sentiment Analysis in Python?](https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/)

You’re probably already familiar with Python, but if not – it is a powerful programming language with an intuitive syntax. Not to mention it’s one of the most popular choices across the data science community, which makes it perfect for our tutorial.

# Step 1: Python Pre-Requisites :

First things first: installing the necessary equipment. You need a Python IDE – I suggest using Jupyter. (If you don’t already have it, follow this [Jupyter](https://365datascience.com/tutorials/python-tutorials/how-to-install-python/) Notebook tutorial to set it up on your device.)

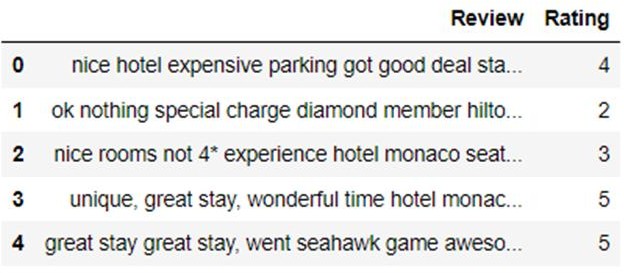
Make sure to have the following libraries installed as well: NumPy, pandas, Matplotlib, seaborn, Regex, and scikit-learn.

# Step 2: Reading the Dataset :

Let’s start by loading the dataset into Python and reading the head of the data frame:

**import** pandas **as** pd

df = pd.read\_csv('tripadvisor\_hotel\_reviews.csv') df.head()



len(df.index) *# 20491*

**Step 3: Data Preprocessing :**

As we already know the TripAdvisor dataset has 2 variables – user reviews and ratings, which range from 1 to 5. We will use “Ratings” to create a new variable called “Sentiment.” In it, we will add 2 categories of sentiment as follows:

* + 0 to 1 will be encoded as -1 as they indicate negative sentiment
  + 3 will be labeled as 0 as it has a neutral sentiment
  + 4 and 5 will be labeled as +1 as they indicate positive sentiment

import numpy as np

def create\_sentiment(rating):

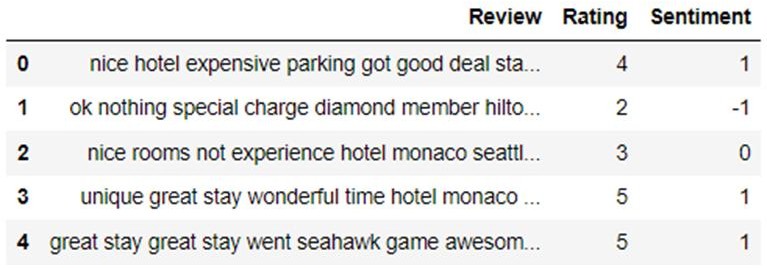
if rating==1 or rating==2:

return -1 # negative sentiment elif rating==4 or rating==5:

return 1 # positive sentiment else:

return 0 # neutral sentiment

df['Sentiment'] = df['Rating'].apply(create\_sentiment)



from sklearn.feature\_extraction.text import re def clean\_data(review):

no\_punc = re.sub(r'[^\w\s]', '', review)

no\_digits = ''.join([i for i in no\_punc if not i.isdigit()]) return(no\_digits)

df['Review'] = df['Review'].apply(clean\_data) df['Review'][0]

**Step 4: TF-IDF Transformation :**

Rather than focusing on individual pieces, inverse document frequency measures how many times a word is repeated across a set of documents. And opposite of the previous metric, here the higher frequency is – the lower the relevance. This helps the algorithm eliminate naturally occurring words such as “a”, “the”, “and”, etc, as they will appear frequently across all documents in a corpus.

Now that you understand how TF-IDF works, let’s use this algorithm to vectorize our data:

from sklearn.feature\_extraction.text import TfidfVectorizer tfidf = TfidfVectorizer(strip\_accents=None, lowercase=False,

preprocessor=None)

X = tfidf.fit\_transform(df['Review'])

**Step 5: Building and Evaluating the Machine Learning Model :**

We can now train our algorithm on the review data to classify its sentiment into 3 categories:

* + Positive
  + Negative
  + Neutral

from sklearn.model\_selection import train\_test\_split y = df['Sentiment'] # target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y)

from sklearn.linear\_model import LogisticRegression lr = LogisticRegression(solver='liblinear') lr.fit(X\_train,y\_train) # fit the model

preds = lr.predict(X\_test) # make predictions

Finally, evaluate the performance:

**from** sklearn.metrics **import** accuracy\_score accuracy\_score

(preds,y\_test) *# 0.86*

Our model has an accuracy of approximately 0.86, which is quite good.

And that concludes our tutorial! For a better understanding of the concept, here is the complete sentiment analysis Python code I’ve used:

import pandas as pd import numpy as np

from sklearn.feature\_extraction.text import re

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression from sklearn.metrics import accuracy\_score

df = pd.read\_csv('tripadvisor\_hotel\_reviews.csv') def create\_sentiment(rating):

res = 0 # neutral sentiment

if rating==1 or rating==2:

res = -1 # negative sentiment elif rating==4 or rating==5:

res = 1 # positive sentiment

return res

df['Sentiment'] = df['Rating'].apply(create\_sentiment) def clean\_data(review):

no\_punc = re.sub(r'[^\w\s]', '', review)

no\_digits = ''.join([i for i in no\_punc if not i.isdigit()])

return(no\_digits)

df['Review'] = df['Review'].apply(clean\_data) tfidf = TfidfVectorizer(strip\_accents=None, lowercase=False,

preprocessor=None)

X = tfidf.fit\_transform(df['Review']) y = df['Sentiment']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y) lr = LogisticRegression(solver='liblinear') lr.fit(X\_train,y\_train)

preds = lr.predict(X\_test) accuracy\_score(preds,y\_test)